Chapter I Number Systems and Codes

I. Introduction

The information (Image, word, number, etc.) processingis carried out by the computer or electronic calculator. The electronic circuits that constitute them can only take two states represented by 0And **1**. So the language used in these machines is called binary number system. **(sequence of 0s and 1s).**

II. Different numbering systems

There are four numberingsystems which are:

- Decimal system;
- Binary system;
- Octalsystem;
- Hexadecimal system.

II.1. Decimal system

It is a system used in the outside world, the numbers used are integers composed of values from 0 to 9.The base of this system is10.

Example

1995:This number is expressed in a system with a base of 10 as follows:

$$1*10^3+ 9 *10^2+ 9 *10^1+ 5 *10^0$$

II.2. Binary system

This system, used by machines (such as computers), consists solely of binary sequences made up of 1s and 0s. Its foundation is based on the number 2.

Example

1001101: The decimal value of this sequence is**:**

$$1*2^6+0*2^5+ 0*2^4+ 1*2^3+ 1*\mathbf{2^2}+0*\mathbf{2}^1+1*2^0=77$$

Each binary digit **(0 or 1)** is called a **bit**, denoted by **b**, and all binary combinations composed of eight bits are called **Bytes**.

Example:11100100

B_0=0, b_1=0, b_2=l, b_3=0, b_4=0, b_5=1, b_6=l, and b_7= 1

Sixteen (16) bits is called a “memory word” or machine word.

Example:1110001010100110

B_0=0, b_1= 1,b_2= 1, b_3= 0,b_4= 0, b_5= 1,..b_{14}=l, andb_{15}=l

II.3. Octal system

The decimal system is the one most used by humans, and the binary system is the one most used in computers.

The binary system has the disadvantage of being difficult for humans to manipulate because the numbers are represented by a sequence of 0s and 1s that are difficult to read or interpret without error. To simplify this, it is more practical to express binary-coded numbers in a more compact form, such as using a system with a base that is a power of 2, which facilitates easy conversion from binary.

In this paragraph we define the octal system, and later the hexadecimal system.

The octal system uses eight symbols to represent numbers, consisting of the digits from 0 to 7. In an octal number, each digit has a weight determined by multiplying the digit's value by 8 to the power corresponding to its position within the number.

Example:N = $(536)_8$

- Octal number	5	3	6
- Power	8^2	8^1	8^0
- Weighting	320	24	6

$320 = 5*8^2$, $24= 3*8^1$, and $6= 6*8^0$

The decimal equivalent of the given octal number is obtained by summing the weights of its digits.

320+24+6 = 350

That is $(536)_8$=$(350)_{10}$

II.4. Hexadecimal system

This system is used on most new digital computers. Its base is equal to 16. There are 16 symbols available to represent a hexadecimal number. These symbols are the ***10 digits*** of the decimal system to which the first 6 letters of the alphabet have been added. The 16 symbols are then: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.**

Each sign of a hexadecimal number has a weighting that is obtained by multiplying the numerical value of the symbol by the power of 16 corresponding to the rank it occupies in the number.

Example:N = $(4CA2)_{16}$

- Number	4	C	A	2
- Power	16^3	16^2	16^1	16^0
- Weightings	16384	3072	160	2

Adding the weights gives the decimal equivalent of the hexadecimal number considered:

16384+3072+160+2 = 19618

Let $(4AC2)_{16}$=$(19618)_{10}$

The interest of these two systems comes from the fact that **8 and 16** have **integer powers of** group of **3 and 4 bits** and therefore, we can **perform conversions**

Octal/Decimal as well as Hexadecimal/Decimal.**(See Table):**

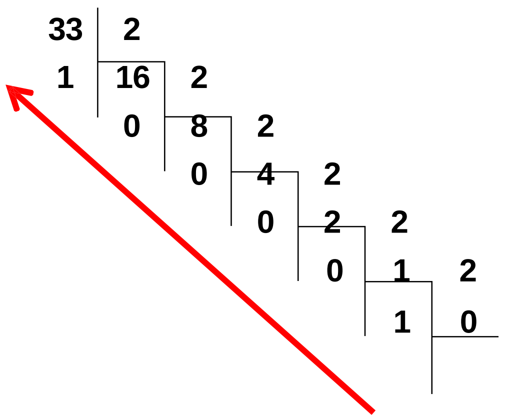
Decimal Number	**Binary Number**	**Octal Number**	**Hexadecimal Number**
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	10001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

III. Transformations (Conversions)

III.1 Decimal / Binary

To convert a decimal number to a binary number, divide the decimal number continuously by the value 2, retaining the sequence of remainders from each division starting from the bottom to the top to obtain the equivalent number.

Example : $(33)_{10}$=$(????)_2$



$(33)_{10}$=$(100001)_2$

If the given number has a fractional part, the conversion process proceeds as follows:

1. Multiply the fractional part of the given number by 2 and keep the integer part of the founded result.
2. Continue multiplying the fractional part obtained from each operation by 2, keep the integer part at each step, until the fractional part becomes zero or repeats. If neither occurs, the result will be an infinite binary sequence, in which case the process can be truncated after a few bits (due to rounding).
3. The final result of the conversion consists of the sequence of integer parts gathered at each step of the multiplication.

Example :

Transform the number 23 into binary?

$(23)_{10}$= $(10111)_2$

Example :

Convert the number (23.625) from base 10 to base 2.

The integer part 23 is already determined, its result is: (23)10= (1011l)2.

For the fractional part 0.625, we proceed as follows:

0.625 * 2 = 1.250

0.250 * 2 = 0.500

0.500 * 2 = 1.00

We see that the last fractional part is zero, so the result will be:

$(0.625)_{10}$= $(0.101)_2$

The final result :$(23.625)_{10}$= $(10111.101\ )_2$

Let us take the infinite case for example $(23.56)_{10}$

For $(23)_{10}$ it is still (10111): but for the fractional part $(0.56)_{10}$, we will have:

0.56 *2=1.12

0.12 *2 = 0.24

0.24*2 = 0.48

0.48 * 2 = 0.96

0.96*2 = 1.92

0.92 * 2= 1.84 etc.,

We note that if we continue the operation of multiplying the fractional part by the value 2, we never reach the stopping criterion for this operation; this means that the binary sequence is infinite. In such cases, the sequence can be truncated by limiting the result to a fixed number of bits..

Hence the final result of: $(23.56)_{10}$ is $(10111.100011\ )_2$.

III.2 Binary/Decimal

To convert a binary sequence into the decimal system, follow these steps:

Sum all the terms in the form of (bit * 2^i), where the summation proceeds from right to left for the integer part and from left to right for the fractional part. The "bit" can be either 0 or 1, and "i" represents the powers of 2. For the integer part, the powers of 2 are positive, starting from 0 up to (n-1), while for the fractional part, the powers of 2 are negative, ranging from (-1) to (-m), where n and m represent the lengths of the binary sequence before and after the decimal point, respectively.

Example :

$(10110)_2 = 0*2^0 + 1 * 2^1 + 1 * 2^2 + 0 *2^3 + 1 * 2^4 = (22)_{10}$

$(10110 . 1011)_2 = ?$

- For the integer part $(10110)_2 = (22)_{10}$
- For the fractional part:

$(0.1011)_2 = (1*2^{-1} + 0*2^{-2} + 1 *2^{-3} + 1* 2^{-4})_{10}$
The final result: $(10110.1011)_2 = (22.6875)_{10}$

III.3 Binary / Octal - Binary / Hexadecimal

The conversion of a binary sequence into the octal system and into the hexadecimal system is done by grouping three and four bits respectively, starting from right to left, each block of bits will be convertible into the requested system.

Example :

$(1010110111)_2 = (?)_8$ and $(?)_{16}$

In base 8 (grouping by blocks of three bits):

$(111)_2 = 7$; $(110)_2 = 6$; $(010)_2 = 2$; $(001)_2 = 1$ From where $(101011011l)_2 = (1267)_8$

- In base 16 (grouping by blocks of four bits),

$(0111)_2 = 7$; $(1011)_2 = B$; $(0010)_2 = 2$ From where $(1010110111)_2 = (2B7)_{16}$

IV. Binary Arithmetic

Binary arithmetic is essential in all types of digital systems. To understand these systems, it is necessary to know the basics of binary addition, subtraction, multiplication, and division.

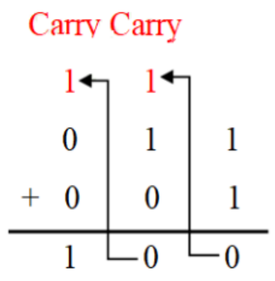
IV.1 Binary Addition

The four basic rules for adding binary digits (bits) are as follows:

- 0 + 0 = 0 Sum of 0 with a carry of 0
- 0+1 =1 Sum of 1 with a carry of 0
- 1+0 =1 Sum of 1 with a carry of 0
- 1 +1=10 Sum of 0 with a carry of 1

Notice that the first three rules result in a single bit and in the fourth rule the addition of two 1s yields a binary two (10).When binary numbers are added, the last condition creates a sum of 0 in a given column and a carry of 1 over to the next column to the left, as illustrated in the following examples:

Example: Add 11+1

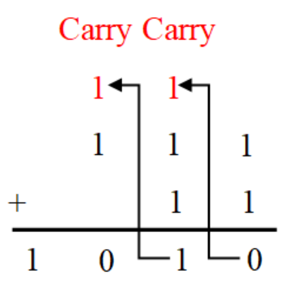


In the right column, 1 + 1 = 0 with a carry of 1 to the next column to the left. In the middle column, 1+1+0=0 with a carry of 1 to the next column to the left. In the left column, 1 + 0 + 0 = 1.

Carry bits

1+0+0= 01	Sum of 1 with a carry of 0
1+1+0= 10	Sum of 0 with a carry of 1
1+0+1= 10	Sum of 0 with a carry of 1
1+1+1= 11	Sum of 1 with a carry of 1

Example: Add 111+11



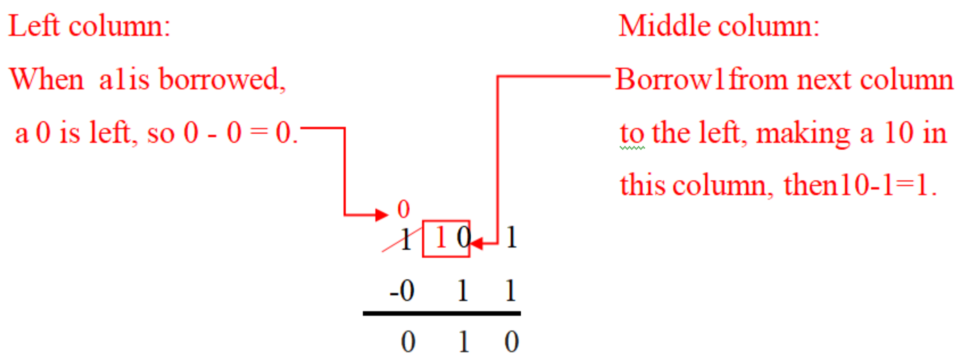
IV.2 Binary Subtraction

The four basic rules for subtracting bits are as follows:

- 0 -0=0
- 1 -1=0
- 1 -0=1
- 10 -1= 1 0-1 with a borrow of 1

When subtracting numbers, you sometimes have to borrow from the next column to the left. A borrow is required in binary only when you try to subtract a 1 from a 0. In this case, when a 1 is borrowed from the next column to the left, a 10 is created in the column being subtracted, and the last of the four basic rules just listed must be applied.

Example: Subtract 011_2 from 101_2.



IV.3 Binary Multiplication

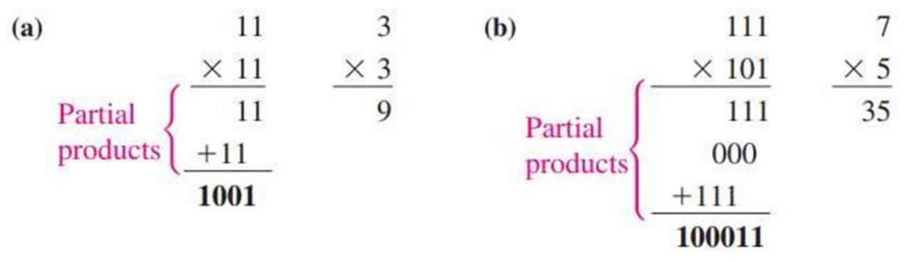
The four basic rules for multiplying bits are as follows:

$0\times0=0,\quad 0\times1=0,\quad 1\times0=0,\quad 1\times1=1$

Multiplication is performed with binary numbers in the same manner as with decimal numbers. It involves forming partial products, shifting each successive Partial product left one place, and then adding all the partial products.

Example: Perform the following binary multiplications:

(a) $11_2 \times 11_2$
(b) $101_2 \times 111_2$



IV.4 Binary Division

The binary division operation is similar to the base 10 decimal system, except the base 2. The division is probably one of the most challenging operations of the basic arithmetic operations.

The binary division is much easier than the decimal division when you remember the following division rules. The main rules of the binary division include:

- 1÷1 = 1
- 1÷0 = Meaningless
- 0÷1 = 0
- 0÷0 = Meaningless

Similar to the decimal number system, the binary division is similar, which follows the four-step process:

- Divide
- Multiply
- Subtract
- Bring down

Important Note: Binary division follows the long division method to find the resultant in an easy way.

Example: 01111100 ÷ 0010

Given

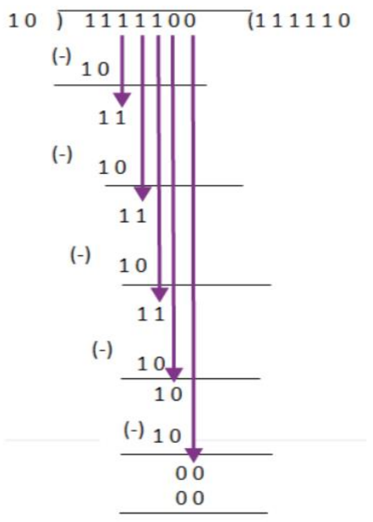
01111100 ÷ 0010

Here the dividend is 01111100, and the divisor is 0010

Remove the zero's in the **Most Significant Bit** in both the dividend and divisor, that doesn't change the value of the number.

So the dividend becomes 1111100, and the divisor becomes 10.

Now, use the long division method.



V. Representation of Negative Binary Numbers

Since binary numbers can have only two symbols either 0 or 1 for each position or bit, so it is not possible to add minus or plus symbols in front of a binary number. We represent negative decimal numbers using a minus symbol in front of them. In computer number representation, these numbers can be distinguishable with the help of an extra bit or flag called **sign bit** or **sign flag** in the Binary number representation system for signed numbers. This sign bit which has a value of sign bit is 0 for positive numbers and 1 for negative binary numbers. The representation of magnitude of positive numbers is easy and does not need any changes. The representation of magnitude of negative numbers is changed accordingly to represent it.

These are: Sign-Magnitude method, 1's Complement method, and 2's complement method.

V.1 Signed Magnitude Method:

We only add an extra sign bit to recognize negative and positive numbers. Sign bit has 1 for negative number and 0 for positive number.

Example:

binary	decimal
00000	0
00010	2
10010	-2
10101	-5
10000	0 (???)

This system is quite good because it's relatively intuitive, but it has two major problems. First, it features two representations for 0 (a "positive" 0 and a "negative" 0), which is not only somewhat confusing but also wastes a representation and complicates hardware checks for 0. Second, the addition method that worked for positive numbers no longer applies to negative numbers. For instance:

```
    0 0 1 0    (2)
+   1 0 1 0   (-2)
  ___________
   1 1 0 0    (-4)   The needed result must to be 0 and not -4
```

V.2 One's Complement method

One's complement of a binary number is another binary number obtained by transforming the 0 bit to 1 and the 1 bit to 0. In the 1's complement format, the positive numbers remain unchanged. The negative numbers are obtained by taking the 1's complement of positive counterparts.

Example:

+9 will be represented as 00001001 in **eight-bit notation**

-9 will be represented as 11110110, which is the 1's complement of 00001001.

In this method, the sign representation has an ambiguous representation of number 0. It means 0 has two different representations one is -0 (e.g., 1 1111 in five bit register) and second is +0 (e.g., 0 0000 in five bit register).

V.3 Two's Complement method

The 2's complement of a binary number is obtained by adding 1 to the 1's complement of the number. In the 2's complement representation, the Most Significant Bit (MSB) indicates the sign: a '0' represents a positive sign, while a '1' represents a negative sign. The remaining bits are used to represent the magnitude. Positive magnitudes are represented in the same way as in sign-bit or 1's complement representation. Negative magnitudes are represented by the 2's complement of their positive equivalents.

Example:

+9 will be represented as 00001001 in **eight-bit notation**

-9 will be represented as 11110111, which is the 1's complement of 00001001.

Note that:
11110111 (represented in 2's complement) = 11110111 (represented in 1's complement) +1

VI. Floating Point representation:

Floating point number are the numbers containing two different part: integer part and the fractional part. It is also refers to real number. A notation called scientific notation is used to represent real numbers in computer systems. Scientific number may be very large or very small. It is also called exponential notation. It takes the form:

$$M*B^e$$

Where **M** is real value called <u>mantissa</u>**,** **B** is the <u>base</u> of the system and, **e** is the integer value called <u>exponent</u> For example:

$325.123 \rightarrow 3.25123*10^2 \rightarrow 0.325123*10^3$

$0.000000245 \rightarrow 0.245*10^{-6} \rightarrow 2.45*10^{-7}$

The binary number can be represented in the scientific notation by keeping base 2.

Ex: 1000.0101 $\rightarrow$ $0.10000101*2^4$ $\rightarrow 1.0000101*2^3$

The binary floating-point format represents numbers by dividing them into three components:

1. **Sign**: This part is always one bit long and indicates the sign of the number.
2. **Exponent**: The exponent is stored using an **excess-N** notation, where the value of N depends on the number of bits allocated for the exponent. For example, if the exponent is stored in 8 bits, the excess-127 method is used, and for 11 bits, the excess-1203 method is applied, and so on.
3. **Mantissa**: The mantissa is stored in normalized form. While floating-point data is commonly represented in a 32-bit format (single precision), and 64-bits.

- The 32 bits format is:

Sign bit	8 bits Exponent	23 bits Mantissa

- The 64 bit format is:

Sign bit	11 bits Exponent	52 bits Mantissa

VII. Binary codes

When numbers, letters, or words are represented by a special group of symbols, this is called encoding, and the group of symbols is called a code. We have seen that decimal numbers can be represented by an equivalent binary number. The group of 0s and 1s in the binary number can be thought of as a code representing the decimal number.

VII.1 Pure binary code

When a decimal number is represented by its equivalent binary number, we call it pure binary code or straight binary coding. As we have seen previously, the conversion between decimal and binary can become long and complicated for large numbers.

VII.2 BCD code

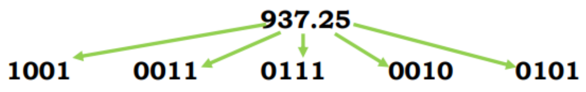
BCD code or Binary Coded Decimal codes. It is a numeric **weighted binary code**, where every digit of a decimal number is expressed by a separate group of 4-bits.

The 8421 code is a type of binary coded decimal (BCD) code. The designation 8421 indicates the **binary weights** of the four bits (2^3, 2^2, 2^1, 2^0). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code.

There are other various BCD codes like, Excee-3, 6311, 2421, 5211, etc. The BCD code is also known as the 8421 code. The following table presents the representations of decimal numbers in various coding systems.

Decimal	BCD (8421)	Excess-3	6311	2421	5211
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0011	0010	0100
3	0011	0110	0100	0011	0101
4	0100	0111	0101	0100	0111
5	0101	1000	0111	0101	1000
6	0110	1001	1000	0110	1001
7	0111	1010	1001	0111	1100
8	1000	1011	1011	1110	1101
9	1001	1100	1100	1111	1111

Example: $(937.25)_{10} = (????)_{BCD}$



$(937.25)10 = (100100110111.00100101)_{BCD}$

VII.3 Gray code

The Gray code is un-weighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that it exhibits only a single bit change from one code word to the next in sequence. Table below is a listing of the four bit gray code for decimal numbers 0 through 15. Notice the single bit change between successive gray code numbers. For instance, in going from decimal 3 to decimal 4, the gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change is in the third bit from the right in the gray code; the other remain the same.

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

- **Binary-to-Gray Conversion**: Conversion between binary code and Gray code is sometimes useful. in the conversion process, the following rules apply:
  - The most significant bit (left-most) in the gray code is the same as the corresponding MSB in binary number.
  - Going from left to right, add each adjacent pair of binary code bits to get the next gray code bit. Discard carry.

Example: The conversion of the binary number 10110 to Gray code is as

$$\begin{tikzcd}
1 \arrow[r, dash, "+"] \arrow[d] & 0 \arrow[r, dash, "+"] \arrow[d] & 1 \arrow[r, dash, "+"] \arrow[d] & 1 \arrow[r, dash, "+"] \arrow[d] & 0 \arrow[d] & \text{Binary} \\
1 & 1 & 1 & 0 & 1 & \text{Gray}
\end{tikzcd}$$

Hence the Gray code of the binary number 10110 is 11101

- **Gray-to-Binary Conversion:** To convert from Gray code to binary, a similar method is used, but there are some differences. The following rules apply:
  - The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.

- Add each binary code bit generated to the gray code bit in the next adjacent positions. Discard carry.

Example: The conversion of the Gray code word 11011 to binary is as follows

$$\begin{tikzcd}
1 \arrow[d] & 1 \arrow[d] & 0 \arrow[d] & 1 \arrow[d] & 1 \arrow[d] & \text{Gray} \\
1 \arrow[ur, "+"] & 0 \arrow[ur, "+"] & 0 \arrow[ur, "+"] & 1 \arrow[ur, "+"] & 0 & \text{Binary}
\end{tikzcd}$$

Hence the binary number of the Gray code word 11011 is 10010

VII.4 Alphanumeric Code

In order to be very useful, a computer must be capable of handling nonnumeric information. In other words, a computer must be able to recognize codes that represent numbers, letters, and special characters. These codes are classified as alphanumeric codes. The most common alphanumeric code, known as the American Standard Code for Information Interchange (ASCII), is used by most minicomputer and microcomputer manufacturers. The ASCII is a seven-bit code in which the decimal digits are represented by the 8421 BCD code preceded by 011. The letters of the alphabet and other symbols and instructions are represented by other code combinations

Example: determine the codes that are entered from the computer's keyboard when the following basic program statement is typed in (**20 PRINT "A=";X**). Also express each in hexadecimal notation.

Character	*ASCII*	*Hexadecimal*
2	0110010	32
0	0110000	30
Space	0100000	20
P	1010000	50
R	1010010	52
I	1001001	49
N	1001110	4E
T	1010100	54
Space	0100000	20
"	0100010	22
A	1000001	41
=	0111101	3D
"	0100010	22
;	0111011	3B
X	1011000	58